MOTOROLA

M68KSYMBG/D2

SYMbug/A and DEbug Monitors Reference Manual

MICROSYSTEMS

QUALITY • PEOPLE • PERFORMANCE

M68KSYMBG/D2

AUGUST 1983

SYMbug/A and DEbug MONITORS

REFERENCE MANUAL

Second Edition

First Edition April 1982

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd) Page

LIST OF TABLES

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

This manual describes the SYMbug/A and DEbug monitor programs, as they are used in the following equipment:

EXORmacs Systems
VMC 68/2 Systems
VERSAmodule 01 and 02 Monoboard Microcomputers (VM01 and VM02)
VMEmodule Monoboard Microcomputers (MVME110)
VME/10 Microcomputer Sytems

Throughout this manual, SYMbug/A is referenced as SYMbug.

1.2 SYMbug/DEbug MONITOR PROGRAMS

The SYMbug and DEbug monitor programs are used to debug other programs whose source code may have been written in Motorola-provided assembly language for execution on the MC68000/MC68010. The language processor, in conjunction with the linkage editor, supplies information to the SYMbug or DEbug monitors.

SYMbug and DEbug must be executed within the VERSAdos operating system.

1.2.1 SYMbug Monitor

SYMbug allows the user to perform the following:

a. Examine, insert, and modify program elements such as instructions, numeric values, and coded information (i.e., data in all its representations and formats).

b. Control execution, including the insertion of breakpoints into a program and requests for breaks or changes in elements of data.

c. Trace execution by displaying information at designated points in a program.

d. Search programs and data for specific elements and sub-elements.

e. Create macro commands allowing user-defined formats and commands.

1.2.1.1 Functional Description. SYMbug is a VERSAdos-resident multitasking utility that allows a user to debug application program(s) in terms close to the actual program itself. Unlike other debuggers that allow only absolute memory access, SYMbug generates and maintains information (assembler symbol names, module names, and section numbers) about the program that is available to the user during debug. SYMbug will automatically equate this type of symbolic information to absolute addresses for the user. Now, it is no longer necessary to reference a current link map to debug a program. Instead, knowledge of module names, symbols, etc., is sufficient to calculate relative offsets and debug the program by reference to an assembler listing. Without the overhead of user-responsible address resolution, the task of debugging a program becomes faster, easier, and reduces the chance of user error.

SYMbug is built around a multitasking kernel. It interfaces with the VERSAdos operating system to provide complete debug control to the user. User interface is via a powerful set of 'primitive' commands. These commands allow the user to:

a. Examine/modify registers and absolute and program-relative memory addresses specified in a number of ways:

   - directly
   - in an expression
   - as an effective address
   - symbolically
   - (also allows control of display/modification formats)

b. Control program execution by allowing the user to:

   - insert breakpoints into the program
   - trace program execution
   - monitor data changes during program execution

c. Direct multitasking functions by allowing the user to:

   - modify task scheduling/information handling
   - modify task attributes/status

d. Expand debugger functions through user generation of:

   - 'macros' built as a series of primitive commands
   - in line command/command block repeat functions
   - default input/output format modifications

e. Access information outside of SYMbug so that the user may save/restore previously defined information:

   - load program(s) from disk
   - save/load symbolic information (macro names/local symbols) to/from disk
   - generate debug session echo (file or printer)

SYMbug is a self-documenting debugger -- that is, the user may utilize the SYMbug HELP command to:

a. Display a brief command syntax summary for all commands.

b. Display a detailed command summary for any command.

1.2.1.2 Operating Environment. In order to ensure proper SYMbug operation conditions, the following complete system is required to run SYMbug:

a. VERSAdos operating system for:

   - on-line management of object code files, symbol table files, and user profile files
   - console I/O and printer output

b. CRT terminal

c. Compatible software to generate/accept the necessary symbolic information required by SYMbug:

   - Assembler Rev. 1.20 (or later)
   - Linker Rev. 1.30 (or later)
   - SYMbug Rev. 2.00
   - SYMbug Help File (:0..SYMBHELP.IN)

1.2.2 DEbug Monitor

DEbug allows the user to perform the following:

a. Examine, insert, and modify program elements such as instructions, numeric values, and coded information (i.e., data in all its representations and formats).

b. Control execution, including the insertion of breakpoints into a program and requests for breaks or changes in elements of data.

c. Trace execution by displaying information at designated points in a program.

1.2.2.1 Functional Description. DEbug is an absolute rather than a symbolic debugger. References to the target program must be absolute addresses or relative offsets from user-defined base registers, rather than relocatable symbols defined at assembly or link time. Thus, there is no debug file as in SYMbug.

DEbug is the functional kernel of SYMbug, and requires less than one-fourth the memory for execution. Macros are not supported, but task level commands and most of the primitive SYMbug commands are available. The self-documenting verb HELP provides a listing of the proper syntax for all supported commands.

While primitive DEbug commands apply only to the prompted foreground task name (initial task loaded), the task level commands take as their first argument the task name to which they apply. If task name is omitted, the foreground task is assumed. Subsequent arguments must be preceded by a comma when task name is omitted. Task level commands are applicable whether the user is in single or multitasking modes. In single task mode, the attach command will not be honored unless preceded by a detach.

1.2.2.2 Operating Environment. The following are required:

a. VERSAdos operating system for:

- I/O support of load modules on-line
- console I/O and printer output

b. CRT terminal

1.3 COMMAND FORMAT

Commands are entered the same as in most other buffer-organized computer systems. A standard input routine controls the system while the user types a line of input. Processing begins only after the carriage return has been entered.

Many primitive commands can be altered by the options field. This provides the user several extensions to the primitive commands.

Several commands are set and reset pairs (set breakpoints, reset breakpoints); rather than having two primitive commands, the form NO is added as the first two characters of the command. For example, the set breakpoint command is BR, and the reset breakpoint command is NOBR.

Command formats are presented in a modified Backus-Naur Form (BNF). Certain symbols in the syntax may be used, where noted, in the real input/output (I/O); however, others are meta-symbols, and their meanings are as follows:

< > The angular brackets enclose a symbol, known as a syntactic variable, that is replaced in a command line by one of a class of symbols it represents.

| This symbol indicates that a choice is to be made. One of several symbols, separated by this symbol, should be selected.

[ ] Square brackets enclose a symbol that is optional. The enclosed symbol may occur zero or one time.

[ ]... Square brackets followed by periods enclose a symbol that is optional/repetitive. The symbol may appear zero or more times.

Operator entries are followed by a carriage return. In examples, operator entries are shown underscored for clarity only.

1.3.1 SYMbug Command Format

The format of a command is:

SYMbug [<task>]? [NO]<command> [<parameters>][;<options>]

where:

SYMbug [<task>]?	Is the SYMbug prompt. <task> is the user foreground task name as displayed by SYMbug.
NO	Performs inverse function of command.
command	Is the command mnemonic.
parameters	Can be expressions or addresses and, when used, are separated by spaces.
options	Multiple options may be selected and, when used, are preceded by a semicolon.

1.3.2 DEbug Command Format

The format of a command is:

Debug [<task>]? [NO]<command> [<parameters>]

where:

Debug [<task>]?	Is the DEbug prompt. <task> is the user foreground task name as displayed by DEbug.
NO	Performs inverse function of command.
command	Is the command mnemonic.
parameters	Can be expressions or addresses and, when used, are separated by spaces.

CHAPTER 2

SYMbug COMMANDS

2.1 INTRODUCTION

This chapter explains how to invoke the SYMbug prompt, describes the command format structure, and provides a detailed description of the primitive and macro commands.

2.2 SYMbug INITIALIZATION

2.2.1 Create Symbol Table For SYMbug

In order to build a symbol table for SYMbug to use, the following steps must be observed in assembling and linking the application program(s):

a. Assemble the program module(s), specifying the debug ('D') option:

=ASM <source>[,<object>[,<listing>]];D

This creates a 'relocatable symbol' file (.RS extension) that contains literal symbol names and their relative offsets for all symbols defined in the source module(s). The assembler creates the relocatable symbol file with the same file field as the source file.

NOTE

Source files should contain an 'IDNT' record. This will correspond with the module name. If IDNT is not used, the module name defaults to file name .RO assembly output file.

b. Link the module(s) assembled in step (a), again specifying the debug ('D') option:

=LINK [<module 1>[/<module 2>]...][,[<load module>][,<listing>]];D

This creates a 'debug' file (.DB extension) that contains the absolute address specifications for the modules as well as the symbol information created during assembly. The linker automatically retrieves the relocatable symbol file(s) corresponding in file field to the relocatable object file name(s) (if the relocatable symbol file(s) exists). The linker gives the debug file the same file field as the first relocatable object module in the link module(s) specification. To allow the user to specify any number of symbols, modules, etc., the linker allocates symbol space dynamically. Therefore, the Z=nn option (which specifies program stack size) may be used to specify a stack size of 'nn' K-bytes. If the linker aborts because of stack overflow, simply invoke it with a larger stack size specification.

c. In order to get SYMbug started, enter:

=SYMBUG [<file name>][.LO]] [<command input to load module>]

where <file name> is the default task to be loaded into SYMbug. If no task is specified, the task(s) should be individually loaded via SYMbug 'LOAD' command(s). SYMbug automatically looks for and loads the corresponding debug file(s) (if it exists).

If SYMbug is to be used as an absolute debugger, there is no need to specify the debug options in the assembly or link phases nor is it necessary to reformat a debug file for SYMbug as there is no need for symbol information. If a module is assembled without debug information, the module name (but no symbols) will be available to SYMbug if a debug file is created during linking/reformatting and SYMbug is used in symbolic mode.

The SYMbug utility will monitor the execution of one to 19 user tasks.

In response to the VERSAdos prompt '=':

=SYMBUG	Invokes the multitasking mode.
=SYMBUG <testprog> [<comline>]	Monitors the named task with an optional command line in single task mode.

The SYMbug start-up sequence includes:

a. Load SYMbug into memory.

b. Display the SYMbug version ID.

c. Prompt for the maximum task count of simultaneous resident tasks for this debugging session. In single task mode, this prompt is suppressed and the named task is automatically loaded and attached.

d. Display the SYMbug prompt.

2.2.2 SYMbug Messages

The following messages may be encountered during execution of SYMbug.

TABLE 2-1. SYMbug Messages

MESSAGE	EXPLANATION
SYMbug <task> ?	<task> is the first four letters of the foreground task name. The user may respond with any primitive or task level SYMbug command.
SYMbug ?	Is the initial prompt in multitask mode. Only LOAD, ATTA, HELP, and QUIT commands are legal until a foreground task is declared.
SYMbug <task> WHAT ?	Signifies a syntax error in the previous SYMbug command. Consult the HELP command listing.
}---> <task> ATTACH	PC=oooooooo. The PC contains the address of the instruction following the instruction which caused the event.
MAP FULL ERROR	There are too many symbols or module names (library modules) for the default symbol table size. To increase the symbol table size: enter SYMBUG with no file name, enter a high maximum task count, and then load and attach to the task. =SYMBUG SYMbug: Revision 2.00 MAXIMUM TASK COUNT (1 -> 19)? 19 SYMbug ? LOAD <file name> SYMbug ? ATTA <file name>
DUP SECT ERROR	When reading in symbol information, the same section or the same module is defined at two different addresses. This could be caused by linking the same relocatable object module twice or linking two modules with the same name.

The following exception event messages will be displayed provided the corresponding bit in that exception mask (XM) is set.

BIT NUMBER	MESSAGE	
0	Not used	
1-15	TRAP #1 -> TRAP #15	PC=xxxxxxxx
16	BUS ERROR	"
17	ADDRESS ERROR	"
18	ILLEGAL INST	"
19	ZERO DIVIDE	"
20	CHK INST	"
21	TRAPV	"
22	PRIV VIOLATION	"
23	LINE 1010	"
24	LINE 1111	"
25-31	Not used	
Unmaskable events	BREAKPOINT	PC=xxxxxxxx
	ATTACHED	"
	DETACHED	"
	STOPPED	"
	TRACE ONE INST	"
	TRACE MAX INST	"
	VALUE CHANGE	"
	VALUE EQUAL	"

These event messages are queued for output as they occur, but will not be displayed until the current SYMbug command has been processed and a new prompt is about to be displayed.

2.2.3 Monitoring Execution of a User Task

All user tasks may be simultaneously active under SYMbug control. There are five ways to initiate a path of execution for a task. Once the command (a through e) has been issued for a task, another command (a through e) may not be entered for the task until execution is terminated by one of the above events.

a. GO - starts the foreground task with OP=0000

b. TR - starts the foreground task with OP=1000 or 0800

c. AS - starts the foreground task with OP=2000 or 3000 and XM=FFFFFFFF

d. STAR - task name starts the named task with OP unchanged

e. STAR ALL - starts all ready or waiting tasks with OP unchanged

NOTE

For a detailed explanation of the OP and XM pseudo registers, refer to paragraph 2.2.9.

Once set into execution, a task will continue to execute until it:

a. Has a normal (maskable) exception that is enabled by its exception mask (XM).

b. Has an unmaskable exception event.

c. Is explicitly stopped by the STOP, TERM, or QUIT command.

d. Terminates normally or abnormally via a TRAP #1 directive under program control.

NOTE: The BREAK key will NOT affect any task's execution.

In multitasking mode, the SYMbug prompt is returned to the user prior to the completion of a task's execution. Any attempt to start execution of a task which is already executing will be ignored. In single tasking mode, the SYMbug prompt is delayed until the event which concludes execution has occurred.

The STAT command will permit a snapshot of a task's progress while in execution. An "e" displayed before a task's status indicates that the task is actively executing. A DF command will then display the register values of the associated task at the time of the last STAT command for a running task. Changes to registers made when a task is not executing, or immediately following execution, are reflected in the DF output. Changes to a task's registers, pseudo registers, or breakpoint addresses are prohibited during execution. To set a new breakpoint, it is necessary to use the following command sequence:

SYMbug	? STOP <task name>	Resets XM to 01FFFFF1.
SYMbug	? TASK <task name>	If not the foreground task.
SYMbug	? BR <address>	Sets the new breakpoint.
SYMbug	? STAR <task name>	Allows previous .OP register options to remain in effect.

The MD command may be used to display a task's memory during execution. MS is also enabled, but extreme caution is urged in regards to dynamically altering an executing task's memory.

2.2.4 Command Entry

SYMbug commands fall into one of two categories; 'primitive' commands and 'macro' commands. Primitive commands are those which are 'resident' (defined) within SYMbug. Macro commands are those which the user defines.

SYMbug accepts free-format command lines. This means that spaces may be embedded anywhere within the line except in the following cases:

a. In a command name

b. In a register, number, or symbol specification

c. In an effective address specification

d. In a macro parameter

e. In a module or section offset specification

A SYMbug command (including any macro parameter expansion) is limited to 79 characters. Command entry is kept as simple and as short as possible to facilitate ease of use. Lowercase letters are not recognized in commands, symbols, or hex numbers. Command name entries are first matched with the user macro table. If no match is found, the SYMbug primitive command table is searched. This process allows the user to redefine a primitive command in other than the default terms.

2.2.4.1 SYMbug Primitive Command List. The SYMbug commands are separated into five groups. They are:

a. Execution group
b. Modify group
c. Display group
d. Session control group
e. Task control group

Group 1: Execution Group

AS	[<address> [<value>[;<mask>]]]	- Address Stop
[NO]BR	[<address>[;<count>]]...	- Breakpoint
G[O]	[<address>]	- Go (Execute)
[NO]IT	<address1> <address2>	- Inside Trace
[NO]OT	<address1> <address2>	- Outside Trace
T[R]	[<count>]	- Trace

Group 2: Modify Group

BF	<address1> <address2> <data>[;<length>]	- Block Fill
BM	<address1> <address2> <address3>	- Block Move
MM	<address>[;<option>]	- Memory Modify
MS	<address> <data>	- Memory Set

Group 3: Display Group

BS	<address1> <address2> <data>	- Block Search
DC	<expression>	- Define Constant or Data Convert
DF		- Display Formatted Registers
MD	<address> [<count>][;<option>]	- Memory Display
OF		- Offset register display

Group 4: Session Control Group

CR	[<count>]	- Command Repeat
DE	[<default option>]	- Defaults
FR	<file name>	- File Read
FS	<file name>	- File Save
HE[LP]	[<command>]	- Display commands
[NO]MA	[<name>]...	- Macro Define
Q[UIT]		- Quit Session
[NO]SD	[<local> [<value>]]	- Symbol Define

Group 5: Task Control Group

```
ATTA  <task name>[,<terminal>|#*]      - Attach task
DETA  [<task name>]                    - Detach task
EVEN  [<task name>],<exception #>      - Event definition
LOAD  <file name>[(command line>]      - Load (task)
MASK  [<task name>],<exception #>      - Mask exception
STAR  [<task name>|ALL]                - Start task(s)
STOP  [<task name>|ALL]                - Stop task(s)
STAT  [<task name>|<status>]           - Status definition
TASK  <task name>[,<note level>]       - Task notify
TERM  <task name>                      - Terminate task
WAIT                                   - Wait task
```

While primitive SYMbug commands apply only to the prompted foreground task name, the task level commands take as their first argument the task name to which they apply. If <task name> is omitted, the foreground is assumed. Subsequent arguments must be preceded by a comma when <task name> is omitted. Task level commands are applicable whether the user is in single or multitasking mode. In single task mode, the ATTA command will not be honored unless preceded by a detach.

In addition, the following keys are of significance during SYMbug execution:

```
BREAK    - Abort command
CTRL-S   - Redisplay line
CTRL-H   - Delete character
CTRL-W   - Hold console output
CTRL-X   - Cancel command line
```

For more information pertaining to these commands, see the appropriate command description(s) which follow this preface.

2.2.4.2 Macro Commands. A 'macro' command is a user-defined command. Macro commands allow the user to redefine/rename primitive commands or to create complex commands built of a string of primitive commands. Macros may also be defined to accept variable parameters for which text is substituted upon invocation.

A macro command may be named and defined in reply to a command request. The format for doing this is as follows:

```
SYMbug     ? MA <macro name>
M= <primitive command>
M= <primitive command>
          .
          .
          .
M=
```

The name of the macro may be one to eight characters long. The definition of the macro command is entered on subsequent lines, with each primitive command followed by a carriage return (CR). To end a macro definition, reply to the prompt (M=) with a (CR) only. When a macro command takes the same name as a primitive command, the macro takes precedence so that the primitive command that it displaces cannot be used except within the body of the new macro.

As an example, Command GO can be redefined by the user to start at a certain address by entering:

```
SYMbug     ? MA GO
M= GO X:MAIN
M=
```

Entering the GO command executes the macro command GO which in turn executes the primitive command GO with the optional starting address. The invocation of GO within the macro will not be recursive, since macros may not contain other macro calls (i.e., they may not be nested). See default GO command function.

The simplest type of macro command consists of a series of constant primitive commands that are to be put into effect in the order written. The definition of the macro command consists in this case of the corresponding string of primitive commands.

Assume that the user finds that a string of commands is frequently entered:

```
MS .D0 $20      - set data register zero to value 20 hex
GO $1000        - start execution at location 1000 hex
WAIT            - wait for event (only in multitasking mode)
MD .A7          - at break, display address register seven
MD $800 $20     - display 20 hex bytes starting at 800 hex
```

To avoid repeatedly entering these five commands, the user may define a macro that will replace the previous series. The user names the macro REPEAT as follows:

```
SYMbug     ? MA REPEAT
M= MS .D0 $20
M= GO $1000
M= WAIT
M= MD .A7
M= MD $800 $20
M=
```

The string of commands is put into effect by invoking the macro by its name REPEAT just as the user would invoke any of the primitive commands. The previously defined macro REPEAT is called as follows:

```
REPEAT
```

The user may set/reset an option to enable/disable listing of the commands in the macro body as they are executed when the macro is invoked. This option need only be set/reset once as follows:

```
SYMbug     ? DE MAL          - to enable macro listing
SYMbug     ? DE NOMAL        - to disable macro listing
```

In many cases, it would be more useful to have a macro in which the parameters and options may be substituted at the time of the macro call. To do this, they may be replaced by symbols consisting of the backslash (\) followed by a digit in the range 0-9. In place of the previous constant macro, the user could define a macro with five parameters denoted by \0, \1, \2, \3, and \4. The macro is entered as follows:

```
SYMbug      ? MA VARIABLE
M= MS \0 \1
M= GO \2
M= WAIT
M= MD \3
M= MD \4 \1
M=
```

NOTE

> SYMbug macro parameters should not be confused with M68000 Family structured assembler macro parameters or chain file substitution arguments.

In order to execute the macro VARIABLE to achieve the same results as executing the macro REPEAT, the user would enter:

```
VARIABLE .D0 $20 $1000 .A7 $800
```

The actual parameters are substituted for the variables. The variables may appear in any numerical order in the body, but the order in which they are entered on the command line is important. (First parameter corresponds to \0, second parameter corresponds to \1, etc.)

Parameters may also be concatenated to other parameters or text by their relative positioning within the macro body as follows:

```
SYMbug      ? MA CONCAT
M= BR \0\1
M= GO \2:MAIN:7
M= MD \3 $20;\4
M=
```

Invoking CONCAT via:

```
CONCAT R:MAIN ;3 X $100 L
```

would generate a macro expansion of the instructions:

```
BR R:MAIN;3
GO X:MAIN:7
MD $100 $20;L
```

This is useful to alleviate the overhead of symbol qualification (see Symbol Resolution) or to add greater flexibility to a macro definition.

When entering a macro, there are three rules to consider:

a. Another macro name cannot be used within the definition of a macro.

b. Syntax of commands entered is not checked until macro is invoked.

c. Macro name must conform to valid symbol syntax for macros (see Symbols).

NOTE

It is not recommended that interactive commands (such as MM) be used inside a macro body. Such commands will interrupt the macro processing flow until the command is completed by the user.

Also, macros and local symbols are defined dynamically from a fixed size table area. There is no preset limit to the number of locals, macros or lines per macro; rather, the user is limited by available space in the table. SYMbug will notify the user when space has been exhausted and no more macros/locals may be entered. Should this happen, the user may delete any number and combination of macro/local symbols to free space for another macro or local symbol. If available table space is exhausted in process of defining macro, the macro is deleted from the table.

2.2.5 Numbers

A number or a string that represents a valid integer constant is said to be either qualified or unqualified, depending on whether or not it is preceded by a prefix which identifies its number base. Qualified constants are prefixed by a term designating the base to be used in evaluating the constant. These prefixes and the four base types they represent are:

```
$ (Dollar)              - Hexadecimal
& (Ampersand)           - Decimal
@ (Commercial "at")     - Octal
% (Percent)             - Binary
```

Unqualified constants take their base from the default input base and, therefore, have no prefix to designate their number base. Upon initialization, SYMbug sets the default input base to hexadecimal; the default input base may be changed via the DE command. A negative value is indicated by preceding any base designation with a unary minus (-).

Example: Default input base is hex

```
DC B        - Yields value of 11 decimal
DC $B       - Yields value of 11 decimal
DC -$B      - Yields value of -11 decimal
DC @16      - Yields value of 14 decimal
DC &21      - Yields value of 21 decimal
DC %100     - Yields value of 4 decimal
```

There is also a special ASCII character that may be used to qualify a term in an expression. This ASCII prefix is a single quote (') preceding a single character.

Example:

```
DC 'A       - Yields value of 65 decimal
DC -'A      - Yields value of -65 decimal
```

2.2.6 Symbols

There are four types of symbols active within SYMbug: assembler, local, macro and task. Assembler, local, and macro symbols may be one to eight characters long; task symbols may be one to four characters long. Any characters entered past these maximum lengths are ignored.

Symbolic address is comprised of three fields delineated by colons:

```
offset:module name:section
```

where:

offset	Is assembler or local symbol, or numeric constants.
module name	Is module name as defined at assembly time.
section	Is a means of qualifying; otherwise, ambiguous references are made to the object module.

An unqualified symbol has no module name or section reference. A partially qualified symbol is offset:module name. A fully qualified symbol is offset:module name:section.

Symbols must obey the following syntax rules:

<assembler symbol> -> (<letter>|.)[(<letter>|$|.|_)]...

<any other symbol> -> <letter>[(<letter>|<digit>)]...

Assembler symbols are those which are created at assembly time and are loaded into SYMbug via the debug file. During SYMbug execution, assembler symbols cannot be created or destroyed; they are active throughout the entire debug session.

NOTE: The term 'assembler symbol' includes module name symbols as well as label symbols.

Local symbols are those which are created by the user during the debug session. They may be equated to any other assembler or previously defined local symbol, relative address, or constant. Once defined, local symbols take on an absolute value (see paragraph 2.2.6.1). These local symbols may also be entered from previously saved symbols on a user profile file. (See FS, FR commands.)

Macro symbols are those which are used to identify macro commands. They are either created during the debug session or read from previously saved macros contained in a user profile file. (See FS, FR commands.)

Task symbols are those which are used to identify named application task(s) for execution control in a multitasking environment. (See Task Control Group commands, paragraph 2.2.4.1.)

Example:

Assembler symbol X exists in module MAIN, Section 0, and in module SUBR, Section 14. MAIN also contains a Section 14. The PC is currently located in module MAIN, Section 0. There is no local symbol X defined.

```
SD R 0:MAIN       - local symbol R = value of relative address 0 in module
                    MAIN, Section 0 (the PC section).
DC X              - value of X in MAIN, Section 0.
DC R              - value of relative address 0 in MAIN, Section 0.
DC X:MAIN         - value of X in MAIN, Section 0.
DC 0:MAIN         - value of relative address 0 in MAIN, Section 0.
DC 0:MAIN:14      - value of relative address 0 in MAIN, Section 0.
DC X:SUBR         - value of X in SUBR, Section 14.
```

Modifying the PC to point to SUBR, Section 14:

```
MS PC 0:SUBR:14
DC X              - value of X in SUBR, Section 14.
DC R              - value of relative address 0 in MAIN, Section 0 (remember
                    that R contains an absolute value; no reference is kept
                    to assembler symbol, relative address, module or section)
DC X:MAIN         - value of X in MAIN, Section 0.
DC X:SUBR         - value of X in SUBR, Section 14.
DC 0:MAIN:0       - value of relative address 0 in MAIN, Section 0.
DC 0:MAIN:14      - value of relative address 0 in MAIN, Section 14.
DC -X:MAIN        - negative value of X in MAIN, Section 0.  (unary minus
                    also allowed for symbols and relative addresses)
```

2.2.6.1 Symbol Resolution. Whenever an 'unqualified' symbol is to be evaluated in an expression, the local symbol table is first searched for a match to determine if the symbol could be a valid local symbol. If the symbol does not exist in the local symbol table, the current module/section (the one to which the program counter is pointing) is searched, then all of the sections of the current module are searched, and finally all sections of all the modules are searched (in order of linkage and section number) for a match. Symbols defined in modules external to the current foreground task are not resolved. If the symbol is not found or if improper syntax is used, the message 'WHAT ?' is displayed.

When the input base is hexadecimal and hex characters are used to designate the offset field of the symbolic address, the offset is resolved as a constant, not as a symbol.

If a change is made from the hex default input base, then any hex numbers used in a string must be prefixed with the hex symbol ($).

Example: Default input base is hex. No user symbol A exists.

```
DC A            - returns value of 10 decimal
SD A 20         - sets local symbol A to a value of 20 hex
DC A            - returns value 10 decimal
DE IN &         - sets default input base to decimal
DC A            - returns value 32 decimal (local symbol A)
DC $A           - returns value 10 decimal ($ prefix --> hex)
```

In order to resolve ambiguities, assembler symbol names and relative addresses may be 'qualified' when entered by also specifying their module name (and section number). This is done as follows:

```
XYZ             - 'unqualified' symbol (may reference local)
XYZ:MAIN        - symbol 'qualified' with resident module
$32:MAIN        - relative address 'qualified' with module
$32:MAIN:14     - relative address 'fully qualified' with module
                  and section number
```

In this manner, symbolic names that appear in more than one module and module names associated with more than one section can be absolutely resolved as to the module/section that the user intended.

The section number qualifier need only be specified for resolution of a relative address within a module name. In other words, if a module MAIN contains Sections 0 and 9, then 0:MAIN could reference either Section 0 or Section 9, depending upon the current PC value, whereas 0:MAIN:0 or 0:MAIN:9 clarifies the relative reference more fully (with respect to section number).

When specifying an assembler symbol offset, any section number qualifier is ignored because of SYMbug's symbol search method and the knowledge that a symbol may not appear in two separate sections of the same module.

Partially qualified symbolic addresses are resolved by first searching the current section in the specified module, and then any section in the specified module.

NOTE

Since unqualified and partially qualified symbols require reference to current PC, they should not be used to reference an executing task because the PC is not fixed.

2.2.7 Expressions

Expressions may contain symbols, constants, or addresses linked together by the following arithmetic operators: add (+), subtract (-), multiply (*) and integer divide (/). There is no hierarchical order of evaluation; rather, the expression is evaluated as it is entered (left to right). No parentheses are allowed to group (sub)-expressions.

Example: MM 3 + 7*4 - is equivalent to 'MM &40' (not 'MM &31') because the addition (not multiplication) is performed first.

Example: Given default input base of hex, consider the following expression:

```
DC -$C +D+    'A/@26* %100+   X:MAIN- R  - &10
```

If 'X:MAIN' is equal to 4, and 'R' is equal to 2, then the expression would yield a result of 4.

Allowing expressions permits the user to specify offsets to memory locations where no symbol exists.

Example: MM XYZ:MAIN +4 - relieves the user of the burden of first calculating the value of address symbol XYZ in module MAIN and then adding the offset.

2.2.8 Registers

In addition to expressions and effective addresses, register names are valid in certain commands as if they were memory locations. The MM, MS, and MD commands will accept register specifications as parameters.

Registers may be from one of four categories -- address, data, offset, or control -- and must conform to the following syntax:

a. <address register> -> .A(0|1|2|3|4|5|6|7)

b. <offset register> -> .R(0|1|2|3|4|5|6|7)

c. <data register> -> .D(0|1|2|3|4|5|6|7)

d. <control register> -> (.PC|.SR|.XM|.OP|.VL|.VA|.VM|.MC)

The stack pointer may be used interchangeably with A7.

Since source register names are equivalent to unqualified hex numbers or unqualified symbols, they must be preceded by a period (.)

Examples:

```
MD .A7        - display contents of address register seven
MD $A7        - display contents of memory location A7 hex
MS .D3 $44    - set data register three to value 44 hex
```

The Exception Mask (XM) Register is unique in that the user can control which halt reasons are enabled through manipulation of this register. When the target is executing and causes an exception, a debug halt will occur if the corresponding bit is set in the exception mask; otherwise, normal processing of that exception will take place (possibly an abnormal termination).

Exception Mask format is four bytes:

Bit Number	Halt Condition
0	N/A
1-15	TRAP #1 - TRAP #15
16	Bus Error
17	Address Error
18	Illegal Instruction
19	Zero Divide
20	CHK Instruction
21	TRAPV
22	Privilege Violation
23	Line 1010
24	Line 1111
25-31	N/A

The mask initially defaults to include all exceptions except TRAP #1, TRAP #2, and TRAP #3. Unless the target task claims break service, the BREAK key is always enabled. If, for example, the user wishes only breakpoint halts (which are affected by planned privilege violations), the exception mask should be set to $40000000. This will also enable halts for non-breakpoint privilege violations.

If the target handles its own zero divide, CHK, and TRAPV exceptions, and the user desires only these halts to be disabled, $1C7FFFF would be the appropriate exception mask.

2.2.9 Pseudo Registers

In addition to the normal task processor registers A0-A7, D0-D7, PC, and SR, the user task can manipulate the following pseudo registers: MC, OP, VA, VL, VM, and XM. These pseudo registers communicate specific constraints to VERSAdos in controlling the user task monitored execution.

MC The maximum count value controls the count of user task instructions to be traced during execution. Displayed as a 4-byte value, the low order two bytes are the count while the upper bytes are updated by VERSAdos to reflect the current count of instructions traced to date. MC is in effect if OP bit 11 is set. Thus OP=2800 combines the address stop and maximum instruction count features.

OP The execution option pseudo register controls the monitored manner of execution. It permits the user task to run free, trace a specified number of instructions, or monitor a given memory address for change in conjunction with the settings of the pseudo registers MC, VA, VL, VM, and XM. Hopefully, the need to adjust these pseudo registers will be minimized by GO, TR, and AS, which provide the most common settings automatically.

VA The value (VA) pseudo register holds a 4-byte value masked by the VM pseudo register for comparison with the current contents at the memory address contained in VL. In the address stop on equal mode setting of OP (=3000) the VALUE EQUAL event will stop the task's execution with the PC at the instruction following that which set the monitored location to the VA value.

VL The value location pseudo register is a 4-byte even (i.e., long word boundary) address of the memory location (long word) within the task's segment to be monitored for change in conjunction with OP=(2000 or 3000). VL through VL+3 must be contiguous RAM locations.

VM The value mask is a 4-byte mask ANDed to the memory content at address location (VL) prior to comparison with the value (VA). Under OP=3000, a VALUE EQUAL event will occur appropriately.

NOTE

A mask setting of $FF000000 will isolate a single even address byte for comparison. A mask setting of $00FF0000 will isolate a single address byte for comparison.

XM The exception mask controls the display of exception event messages declared previously. The mask initially defaults to 01FFFFF1, which enables all events except TRAPS #1, #2, and #3. If a user task desired to handle its own zero divide exceptions, a mask value of 01F7FFF1 would apply.

2.2.10 Addressing Modes

A problem that is common in debugging is user control/modification/display of a stack operation. In non-symbolic debugging, the user must first interrogate the address register used as the stack pointer to obtain the current stack location. Then the user must calculate the offset desired, add this offset to the register value, interrogate and add the value of any index register, and then perform a memory modify on the resultant value location. In symbolic debugging, the user may enter the desired stack location as an 'effective address'.

Example:

Assume that the stack pointer is in address register A7, the stack area in question is at a static displacement of -10 hex, and data register D3 is being used to calculate dynamic offsets from the stack area. The user wishes to examine the current stack location. In non-symbolic terms, this is accomplished by:

a. MD .A7 - to get the stack pointer location

b. MD .D3 - to get the index register offset

c. MD -$10 + x - where 'x' is the sum of (A7) + (D3) to reference current stack location.

In symbolic terms, the 3-step operation becomes simply:

MD -10(A7,D3) - performs the above operations without user-introduced errors due to arithmetic, or register examination errors.

Other examples of effective address capabilities of SYMbug are:

(An)	- Address Register Indirect
(An,Rn)	- Address Register Indirect with Index
m(An)	- Address Register Indirect with Displacement
m(An,Rn)	- Address Register Indirect with Index and Displacement (.W suffix not allowed)
(PC)	- Program Counter Indirect
(PC,Rn)	- Program Counter Indirect with Index
m(PC)	- Program Counter Indirect with Displacement
m(PC,Rn)	- Program Counter Indirect with Index and Displacement

where 'n' is a valid digit in the range 0-7, 'm' is any valid integer, and 'R' is an address (A) or data (D) register specification.

NOTE

For PC relative, 'n' is an absolute offset, not the destination address as used in resident assembler syntax.

Effective addresses are valid anywhere that expressions are valid as a command operand. However, effective addresses are not allowed in expressions.

2.2.11 Options

There are eight groups of options available to the user for control of I/O operations in SYMbug. They are:

GROUP 1: Output Length

B	- Byte (8-bits)
W	- Word (16-bits)
L	- Long Word (32-bits)

GROUP 2: Output Type

$	- Hexadecimal
&	- Decimal
@	- Octal
%	- Binary
'	- ASCII Character
I	- Integer
DI	- Disassembly

GROUP 3: Secondary Output Device

#PR	- Printer
#FN	- Disk File

GROUP 4: Macro Control

MAL	- Enable Macro Expansion
NOMAL	- Disable Macro Expansion

GROUP 5: Breakpoint Control

BRE	- Enable Breakpoint Logic
NOBRE	- Disable Breakpoint Logic

GROUP 6: Default Input Base

IN $	- Default Input Base Hexadecimal
IN &	- Default Input Base Decimal
IN @	- Default Input Base Octal
IN %	- Default Input Base Binary

GROUP 7: User File

FILE <file name> - Secondary File Output Link to External Disk File

Group 8: Output Echo Control

ECHO	- Enable Output Echo
NOECHO	- Disable Output Echo

Options in groups 1, 2, and 3 may be entered on the command line of certain primitive commands. All of the options may be specified via the DE command. Some options may override others, e.g., the DI option will override the length option.

The default options are:

Option Group	Default	Value
1	B	Byte output length is selected
2	$	Hex output type is selected
3	#PR	Printer output
4	MAL	Macro expansion is enabled
5	BRE	Breakpoint logic is enabled
6	IN $	Default input base is hexadecimal
7		
8	NOECHO	Secondary echo is disabled

2.2.12 Attaching/Detaching Secondary Echo Device (via DE Command)

In this section, the term 'attach' will correspond to the command DE ECHO, and the term 'detach' will correspond to the command DE NOECHO.

2.2.12.1 Secondary Output to Printer. A printer may be assigned as the secondary output device as follows:

```
SYMbug    ? DE #PR (assign printer device)
SYMbug    ? DE ECHO (attach printer)
```

2.2.12.2 Secondary Output to Disk File. A disk file may be assigned as the secondary output device as follows:

```
SYMbug    ? DE #FN  (select file echo)
SYMbug    ? DE FILE <file name> (assign file)
SYMbug    ? DE ECHO (attach file)
```

or

```
SYMbug    ? DE FILE <file name> (assign file)
SYMbug    ? DE #FN  (select file echo)
SYMbug    ? DE ECHO (attach file)
```

The DE ECHO command must follow file assignment and selection default (DE) commands. If the DE ECHO command is given and the <file name> has not been previously assigned (in this debugging session), the message 'ATTACH NOT ALLOWED' will be displayed.

The first attach to a <file name> for secondary output will overwrite the file. The file must already exist on disk before the attach is attempted. Subsequent detach/attach command series (to the same file) will append echo output to the end of the file.

The user is cautioned that any time the secondary output device designation is to be changed from a previous assignment, a DE NOECHO command must be given before the echo device is changed. The logical sequence then is:

```
SYMbug    ? DE NOECHO     (detach current echo device; not needed if first
                           selection)
SYMbug    ? DE <device>   (#PR, #FN)
SYMbug    ? DE ECHO       (attach to <device>; if <device> is #FN, assign
                           file first)
```

This also applies to changing to a new <file name> (even if file device was previously selected as echo device) as follows:

```
SYMbug    ? DE NOECHO                (detach current echo device)
SYMbug    ? DE #FN                   (not needed if #FN is current echo device)
SYMbug    ? DE FILE <file name>      (select new file designation)
SYMbug    ? DE ECHO                  (attach file)
```

Issuing a DE FILE <file name> command, which specifies a <file name> which had previously attached, will overwrite the file on the next attach (any previous echo information will be lost). Also, the DE <device> command is not needed unless the <device> to be selected is different from the <device> which is currently selected.

NOTE

When using the secondary output to a file, SYMbug uses the NULL catalog for the file name, not the default user catalog.

2.3 PRIMITIVE COMMANDS

SYMbug primitive commands are listed in the following table.

TABLE 2-2. SYMbug Primitive Commands

COMMAND SYNTAX	DESCRIPTION
AS [<address> [<value>[;<mask>]]]	Address stop
BF <address1> <address2> <data>[;<length>]	Block fill
BM <address1> <address2> <address3>	Block move
[NO]BR [<address>[;<count>]]...	Set/reset breakpoint
BS <address1> <address2> <data>	Block search
CR [<count>]	Command repeat
DC <expression>	Define constant or Data convert
DE [<default option>]	Defaults
DF	Display formatted registers
FR <file name>	File read
FS <file name>	File save
G[O] [<address>]	Go (execute)
HE[LP] [<command>]	Display commands
[NO]IT <address1> <address2>	Set/reset inside trace
[NO]MA [<name>]...	Set/reset macro define
MD <address> [<count>][;<option>]	Memory display
MM <address>[;<option>]	Memory modify
MS <address> <data>	Memory set
OF	Display Offset register
[NO]OT <address1> <address2>	Set/reset outside trace
Q[UIT]	Quit (go to VERSAdos)
[NO]SD [<local> [<value>]]	Set/reset symbol define
T[R] [<count>]	Trace
ATTA <task name>[,<terminal>|#*]	Attach task
DETA [<task name>]	Detach task
EVEN [<task name>],<exception #>	Event definition
LOAD <file> [<command line>]	Load (task)
MASK [<task name>],<exception #>	Mask exception
STAR [<task name>|ALL]	Start task(s)
STAT [<task name>,<status>]	Status definition
STOP [<task name>|ALL]	Stop task(s)
TASK <task name>[,<note level>]	Task notify
TERM <task name>	Terminate task
WAIT	Wait task
BREAK	Abort command
CTRL-S	Redisplay line
CTRL-H	Delete character
CTRL-W	Suspend output (1)
CTRL-X	Cancel command line
CR (Carriage Return)	Send line for execution

NOTE:

(1) When CTRL-W is used, the entry of any character will cause the output display to continue.

In SYMbug command syntax, <address> arguments are resolved to absolute memory locations and may take any of the following forms:

<address>	Examples
A numeric constant	4AB2 or &1000
A symbolic constant	START (a local symbol)
A symbolic address	4A:MAIN:9 or HERE (an assembler symbol)
An effective address	(PC) or 6(A5,D3)
An expression	8+R4 or HERE-$100

For the commands MD and MS, the scope of <address> is enlarged to accept register forms as follows:

<address>	Examples
A register	.A3 or .D6
An offset or pseudo register	.R3 or .VL or .OP

2.3.1 Address Stop (AS)

AS [<address> [<value>[;<mask>]]]

where:

address — Is the memory location specified for test (any address mode). A 4-byte comparison is made according to the mask value.

value — Is the pattern to be compared for match at <address> (number/local symbol). If <value> is given, execution is in the 'stop-on-address-equal' mode -- otherwise, in the 'stop-on-address-change' mode.

mask — Is the mask value for comparison (B, W, L):

- B Sets a mask of $FF000000 (MSB).
- W Sets a mask of $FFFF0000 (MSW).
- L Sets a mask of $FFFFFFFF (exact). Default mask is $FFFFFFFF.

The AS command allows the user to set/display a breakpoint type condition in RAM user memory. Whenever a specified pattern is written to the memory address in question, an exception event occurs.

When the AS command is used with parameters, target task begins execution. When used with no parameters, the current AS set-up is displayed. Multiple address stop conditions are not allowed. The .OP register is set to $200 unless a value is supplied which sets .OP to $300 (stop-on-address equal mode).

EXAMPLES		COMMENTS
SYMbug	? AS $100 $FE	Specifies address stop condition when memory contents at location $100-$103 become equal to $000000FE.
SYMbug	? AS $100	Specifies address stop whenever locations $100-$103 are changed from their current value.

See also: GO, TR

2.3.2 Block Fill (BF)

BF

BF <address1> <address2> <data>[;length]

where:

address1	Is the lower limit for fill operation.
address2	Is the upper limit for fill operation.
data	Is the fill pattern (number/local symbol) and must be a numeric or symbolic constant.
length	Is the size of repeat pattern (B, W, L). Default is L. If <address> is odd boundary, any <length> specification must be 'B' (default for odd boundary).

The BF command allows the user to repeat a specific pattern throughout a determined user memory range.

The BF command fills according to <length> and only to upper boundary regardless of data length.

EXAMPLES		COMMENTS
SYMbug	? BF $101 $200 $FF;B	Fill each byte of memory from location $101-$200, inclusive with the pattern $FF.
SYMbug	? BF $30 $6F $FF;L	Fill each long word of memory from location $30-$6F with the pattern $000000FF (value is right justified for long word).
SYMbug	? BF 4 4 $FF	Sets location 4 to $00. (Long word is $000000FF.)
SYMbug	? BF 4 4 $FF;B	Sets location 4 to $FF.

BM <address1> <address2> <address3>

where:

address1	Is the lower limit of block to be moved. Must be same parity as <address3> (even/odd).
address2	Is the upper limit of block to be moved.
address3	Is the block relocation target address. Must be same parity as <address1> (even/odd).

The BM command allows the user to relocate a memory block. The move is non-destructive in that moving a block to an address within the block will not destroy the integrity of the moved data. The block move is byte oriented.

EXAMPLES		COMMENTS
SYMbug	? BM $100 $200 $300	Relocate memory block in range $100-$200 to memory at $300-$400.
SYMbug	? BM $0 $10 $2	Relocate memory block in range $0-$10 to $2-$12. $2-$12 will accurately represent the former data that was in $0-$10.
SYMbug	? BM DATA DATA+5 4:MAIN:0	Relocate memory block DATA to the symbolic address in section zero in module MAIN.

2.3.4 Set Breakpoints (BR) BR
Reset Breakpoints (NOBR) NOBR

```
BR  [<address>[;<count>]]...
NOBR  [<address>[;<count>]]...
```

where:

address — Is the program location where breakpoint is to occur (any even address).

count — Is the number of times the instruction at <address> is executed to access <address> before program breakpoint is to occur (decimal number > 0). The <count> may be 1-65535 (must be qualified decimal). Default is 1 (assumed decimal). Do not prefix & to constant.

The BR and NOBR commands set/reset breakpoints at user program locations. Each breakpoint may be specified with a count to enable program loops. Up to 8 breakpoints may be entered at a time, with a maximum of 10. SYMbug will distinguish between exception event 22 caused by breakpoints and user-generated privilege violations (refer to paragraph 2.2.2).

SYMbug affects breakpoints by implanting a privileged instruction at the breakpoint address. Therefore, breakpoints will not be properly processed unless bit 22 is set in the exception mask.

Rather than removing and re-entering breakpoints with BR and NOBR, it may be convenient to temporarily disable them with the default option DE NOBR.

EXAMPLES		COMMENTS
SYMbug	? BR X	Sets breakpoint at address specified by symbol X (default <count>=1) (X may be a local or assembler symbol).
SYMbug	? BR $30;4	Sets breakpoint at address $30 with <count>=4.
SYMbug	? BR $30;2 $100 $50;7	Sets 3 breakpoints in user program.
SYMbug	? NOBR $30;1 $50	Removes breakpoints from location $30 and $50 (<count> is ignored on remove option).
SYMbug	? NOBR	Removes all breakpoints from program (if they exist).

BS <address1> <address2> <data>

where:

address1	Is the lower limit for search operation.
address2	Is the upper limit for search operation.
data	Is the pattern to be searched for (number/local symbol or ASCII string).

The BS command allows the user to scan a specified memory range for a certain pattern. Each address where the pattern occurs is displayed for the user. The length of <data> is determined by the length of the constant entered. Thus, $FF will search bytes and $FFFF30 will search for a hex string of three adjacent bytes. Searches in byte increments through range but updates past a found string before looking for next occurrence; e.g., looking for $FFFFFF in a 16-byte string of $FF's will encounter matches at offsets 0, 3, 6, 9, and 12.

EXAMPLES		COMMENTS
SYMbug	? BS $100 $200 'ABC'	Search memory from $100-$200, inclusive, for the pattern $414243 ('ABC').
SYMbug	? BS $0 $100 $FF	Search memory from $0-$100 for the pattern $FF.

CR [<count>]

where <count> is the number of times to invoke next command (decimal number >0). The <count> may be 1-65535 (do not prefix & to the constant). Default <count> is 1.

The CR command allows the user to specify multiple invocations of the command which follows it. This permits the user to build primitive or macro command loops. The next primitive/macro command is repeated <count> times. The macro being repeated may itself contain a CR command (remember that macro's may not call other macro's).

EXAMPLES		COMMENT
SYMbug	? CR 2	
SYMbug	? ABC	Invokes macro 'ABC' twice.

DC <expression>

where <expression> is the valid arithmetic expression.

The DC command allows the user to resolve arithmetic expression and/or symbolic values. The value is returned in symbolic, hex, decimal, and binary formats.

EXAMPLES		COMMENTS
SYMbug	? DC CAT+DOG	Returns value of sum of assembler/local symbols 'CAT' and 'DOG' (if they exist). A syntax error occurs if an undefined symbolic constant is referenced.
SYMbug	? DC $15+@17+CAT+%11+&21/&33*DOG	If 'CAT' contains value 6 and 'DOG' contains value 3, returns value 6.
SYMbug	? DC 0:MAIN:3	Returns absolute address corresponding to logical address 0 in assembler module 'MAIN', section 3.
SYMbug	? DC X:MAIN	Returns absolute address corresponding to logical address associated with assembler symbol 'X' in assembler module 'MAIN'.

DE [<default option>]

where <default option> is the specific default option to be changed:

[NO]MAL	macro expansion
[NO]ECHO	secondary output enable
[NO]BR	breakpoint enable
IN ($|&|@|%)	input base
($|&|etc.)	output type (refer to option group 2, paragraph 2.2.11)
B, W, L	output length
#PR, #FN	secondary output device
FILE <file name>	secondary output file

The DE command allows the user to examine/modify I/O and control specifications pertaining to the debug session.

EXAMPLES		COMMENTS
SYMbug	? DE IN &	Modifies default input base to decimal. Any unqualified numbers will now be evaluated as decimals.
SYMbug	? DE NOMAL	Disables macro expansion option so that macro subcommands will not be listed as they are invoked.
SYMbug	? DE	List current defaults.

MAL NOECHO NOBRE BIN IN HEX OUT;L #PR FILE=

Macro expansion enabled, echo disabled, breakpoints disabled, binary default input base, hex default output, long default length, printer echo device, no file.

2.3.9 Display Formatted Registers (DF)

DF

The DF command displays the data, address, program counter and status registers of the foreground task.

EXAMPLE	COMMENT
SYMbug ? DF	Display D0-D7, A0-A7, PC, SR

```
D0-D7  01000000 00000000 000A02CD 05050505 00000000 FFFFFFFF ...
A0-A7  00000AAE 00000AD2 00000BCC 00001000 00001020 00000A00 ...
PC=00010000   SR=0000
```

See also: MD, MS

FR <file name>

where <file name> is a valid VERSAdos file name (no default extension).

The FR command allows the user to recall saved macro/local symbol definitions from a previous debug session. Any current macro's and local symbols will be destroyed. Reading a file that was not previously created/modified via an FS (file save) command will generate unpredictable results.

EXAMPLE		COMMENT
SYMbug	? FR SYMBUG.PF	Restores macro/local symbols from file SYMBUG.PF for current debug session.

See also: FS, MA, SD

FS <file name>

where <file name> is a valid VERSAdos file name (no default extension).

The FS command allows the user to save any macro and local symbol definitions to the specified disk file for later use. If file exists, it will be overwritten; otherwise, it will be created for the user.

EXAMPLE		COMMENT
SYMbug	? FS SYMBUG.PF	Saves defined macros/local symbols to file SYMBUG.PF.

See also: FR, MA, SD

G[O] [<address>]

where <address> is the starting address to locate PC before execution begins (any address mode; must specify even address). The <address> must be on even (word) boundary.

The G or GO command allows the user to initiate target program execution in free run mode. The user may optionally specify a starting address where execution is to begin. Execution starts at current PC unless <address> is present. Only a breakpoint, address stop, STOP command, or execution error will cause the program to terminate execution. The GO command also sets the .OP address to $0000.

EXAMPLES		COMMENTS
SYMbug	? GO	Begin program execution at current PC location.
SYMbug	? GO $100	Begin program execution at location $100.

See also: AS, BR, STOP

HE[LP] [<command>]

where <command> is any valid SYMbug primitive command.

The HE or HELP command allows the user to get an abbreviated or detailed list of SYMbug commands.

EXAMPLES		COMMENTS
SYMbug	? HE	Displays a list of all SYMbug primitive commands and their syntax.
SYMbug	? HELP AS	Displays a more detailed description of the SYMbug AS command.

2.3.14 Define Trace (IT) / Delete Inside Trace (NOIT)

IT
NOIT

```
IT <address1> <address2>
NOIT
```

where:

address1	Is the lower boundary of trace range.
address2	Is the upper boundary of trace range.

The IT command allows the user to specify an address range to be used in conjunction with the TR command. If PC address is within the range after the trace, it is reported to the user. After tracing, if the PC address is outside the trace boundary limits, the event message TRACE ONE INST or TRACE MAX INST is suppressed.

Only one address range for trace inclusion will be in effect. Each subsequent IT command replaces any existing trace boundaries.

The NOIT command allows the user to remove the inside trace range specification. The NOIT command has no address arguments.

EXAMPLES		COMMENTS
SYMbug	? IT $0 $100	Specifies that if final PC trace address falls within the range $0-$100, it is to be reported to user.
SYMbug	? NOIT	Remove current inside trace range specifications.

See also: OT, TR

2.3.15 Macro Define and Display (MA) MA
Macro Delete (NOMA) NOMA

MA [<name>]...
NOMA [<name>]...

where <name> is a valid symbol name (1-8 alphanumeric characters).

The MA command allows the user to define/delete a complex command consisting of any number of SYMbug primitive commands with optional parameter specifications.

In response to the macro definition prompt M=, enter a SYMbug command, including a carriage return. The prompt M= will appear for the next command to be included in the macro. Commands entered are not checked for syntax until the macro is invoked. Exit definition mode of a macro by a carriage return (null line). The macro may now be listed by typing MA <name>. If the macro contains errors, it must be deleted (NOMA) and redefined. A macro containing no primitive SYMbug commands is deleted from the macro table. When the macro table is full, the user will be exited automatically from definition mode. Instead of the M= prompt, the syntax error prompt 'SYMbug <task> WHAT ?' will appear. Type MA to display all currently defined macros. Delete unneeded macros with NOMA command or use FS command to save all macros and local symbols to a disk file, then re-enter the macro definition mode. SYMbug commands contained in macros may reference arguments supplied at invocation time. ARGUE ZERO 1 TOO would invoke the macro named ARGUE. The text strings 'ZERO', '1', and 'TOO' would replace references to \0, \1, and \2 within the body of the macro.

If the named macro exists, it is displayed; otherwise, a prompt of M= is displayed, indicating macro definition mode. If more than one <name> specified, display all <name>s. If any of the <name>s are not defined, SYMbug generates a syntax error. If deleting with NOMA, and <name> is not in table, SYMbug generates a syntax error and any subsequent <name>s are not deleted.

EXAMPLES		COMMENTS
SYMbug M=MD $0 M=GO\0 M=	? MA ABC	Defines macro 'ABC' to display memory at location $0 and then start execution at address specified in parameter 0. '?ABC $100' would be a valid invocation of the new macro. The string '$100' would be substituted for the string '\0' on echo to user if the macro expansion option is enabled.
SYMbug	? NOMA ABC	Delete macro 'ABC' from user macro table.
SYMbug	? MA	Display all macros.
SYMbug	? NOMA	Delete all macros from user macro table.

See also: DE, FR, FS

MD <address> [<count>][;<option>]

where:

address	Is the starting memory location for display.
count	Is the optional number of bytes to display (decimal number > 0). The default <count> is 16 bytes.
option	Is DI for disassembly format.

The MD command allows the user to examine a variably sized block of user memory or task register. Regardless of actual <count>, the display will always be an integral of 16 bytes of hex data. This enables the user to display task registers (including pseudo registers) individually.

EXAMPLES		COMMENTS
SYMbug	? MD $100	Displays 16 bytes (default <count>) of memory starting at location $100.
SYMbug	? MD $101 $17	Displays 32 bytes (integral number of 16 bytes displayed) starting at location $101.
SYMbug	? MD .OP	Display user task OP pseudo register.
SYMbug	? MD 6:MAIN;DI	Displays 16 instructions in disassembly format beginning at the absolute address equivalent to 6:MAIN.
SYMbug	? MD .D1	Displays data register D1.
SYMbug	? MD D1	Displays 16 bytes of data at address D1.
SYMbug	? MD START+16 4;DI	Displays four instructions in disassembly format beginning at the absolute address equivalent to 16 above the relocated assembler symbol.

MM <address>[;<option>]

where:

address	Is the memory location to start display/modify.
option	Is DI for assembler syntax input.

The MM command allows the user to examine/modify user memory locations in an interactive manner.

'DI' option will perform disassembly of data and accept modifications in assembler syntax (absolute values in disassembly will be resolved to symbols). Depress (CR) to disassemble the next instruction. To replace an instruction, type space, then enter assembler mnemonic and operand fields. Assembler syntax errors are indicated by an X under the offending field. Acceptable assembler syntax is displayed automatically along with the next disassembled instructions.

The line terminators for data entry are: ([data] not valid with 'DI')

- [data] (CR) - [update and] step to next location
- [data] ^ - [update and] step back (not valid with 'DI')
- [data] . - [update and] terminate 'MM'
- [data] = - [update and] re-open location

EXAMPLES		COMMENTS
SYMbug	? MM $100;L	Display/modify memory starting at location $100. I/O will be in groups of 16 bits (long word).
SYMbug	? MM $100	Display/modify memory starting at location $100. (Default I/O is hexadecimal bytes.)
SYMbug	? MM START;DI	Disassembles and displays the instruction at address START. Depress (CR) to step to next instruction.

See also: MS

MS <address> <data>

where:

address — Is the register or user memory location.

data — Is the new content of address. It may be an ASCII string enclosed in quotes, or a succession of numeric byte values each delimited by spaces.

The MS command allows the user to store a value into a specified memory location or register. Only significant bytes of new <data> are entered into memory. Any register (including pseudo registers) may be set.

EXAMPLES		COMMENTS
SYMbug	? MS .D0 $100	Set data register zero to value $100.
SYMbug	? MS $100 'ABC'	Set memory locations $100-$102 to $41, $42, $43 respectively ('A', 'B', and 'C').
SYMbug	? MS $0 $FFF	Set memory locations $0-$1 to $0F, $FF respectively.
SYMbug	? MS $10 0 'ABC' $10	Set memory locations $10-$14 to $00, $41, $42, $43, $10 respectively.
SYMbug	? MS $50 $00FF 0 $FF	Set memory locations $50-$52 to $FF, $00, $FF respectively.

See also: MM, BF

OF

Offset registers establish base addresses usable in expressions.

If assembler symbols are available from a .DB file, offset registers are not so valuable. When no .DB file is available, base registers may be loaded with absolute addresses of assembler sections, facilitating offset references matching the assembler listing (e.g., if an object module began at $1A00 and if .R1 were initialized with the SYMbug MS command to $1A00, then to disassemble the first 16 executable instructions, type MD0+R1;DI.

The OF command displays all user-defined offset registers numbered .R0 through .R7.

EXAMPLES		COMMENTS
SYMbug	? OF	Displays the offset registers.
R0-R7	00000000 00000020 00010000 00000000 00000000 ...	
SYMbug	? MS .R1 1000	Sets offset register one to $1000.

2.3.20 Set Outside Trace (OT) / Delete Outside Trace (NOOT)

OT
NOOT

```
OT <address1> <address2>
NOOT
```

where:

address1	Is the lower boundary of trace exclusion.
address2	Is the upper boundary of trace exclusion.

The OT command allows the user to specify an address range to be used in conjunction with the TR command. If PC address after trace is outside the range, it is reported to the user; otherwise, the event message is suppressed. Only one address range for trace exclusion will be in effect. Each subsequent OT command replaces any existing address boundaries.

The NOOT command has no arguments. It removes existing trace limits.

EXAMPLES		COMMENTS
SYMbug	? OT $100 $200	Specifies that if the final PC address falls outside the range $100-$200, it is to be reported to the user.
SYMbug	? NOOT	Remove current outside trace range specifications.

See also: TR, TT

2.3.21 Terminate Debugging Session (Q) Q

Q[UIT]

The Q or QUIT command allows the user to terminate the current debug session. SYMbug and all of the user tasks attached to SYMbug are terminated.

EXAMPLE		COMMENT
SYMbug =	? QUIT	The current debugging session is terminated, and control is returned to VERSAdos.

2.3.22 Symbol Define (SD) SD
Symbol Delete (NOSD) NOSD

```
SD [<local> [<value>]]
NOSD [<local>]
```

where:

local	Is a valid symbol name (1-8 alphanumeric characters).
value	Is the value to be associated with <local> (any address mode).

The SD command allows the user to define/examine/modify user local symbols. It also redefines a local symbol without first deleting it. When out of room in the table, the symbol is not entered. The FS command saves local symbols and macros which share a common table.

The NOSD command allows the user to delete specified or all local symbols from the local symbol table.

EXAMPLES		COMMENTS
SYMbug	? SD X 0:MAIN:14	Defines local symbol X with value of relative address 0 in module MAIN, Section 14 ($27).
SYMbug	? SD Y X+1	Defines local symbol Y with value of local symbol X + 1.
SYMbug	? SD X $77	Redefines local symbol X to take on value $77.
SYMbug X = $	? SD X 77	Returns current value of local symbol X.
SYMbug	? NOSD X	Deletes local symbol X from local symbol table.
SYMbug Y = $	? SD 27	Displays all current local symbols.

See also: FR, FS

T[R] [<count>]

where <count> is the number of instructions to execute before forcing program breakpoint (decimal number > 0). The <count> may be 1-65535 (must be unqualified decimal). Default <count> is 1.

The TR command allows the user to monitor program execution on an instruction by instruction level. The user may optionally execute several instructions at a time. Execution starts at current PC. The PC displayed with the event message is of the next instruction to be executed. In 'trace <count>' mode, TRAP instructions are not counted. The TR command also sets the .OP register to $1000.

EXAMPLES		COMMENTS
SYMbug	? TR 7	Executes seven instructions starting at current PC and then returns control to debugger.
SYMbug	? TR 1	Execute one instruction.
SYMbug	? TR	Same as above (default <count>=1).

See also: IT, OT

2.3.24 Attach a Task to SYMbug (ATTA)

ATTA <task name>[,#<terminal>|#*]

The ATTA command will attach SYMbug to a task already in memory. Tasks usually are first loaded using the LOAD command; however, they may be attached to SYMbug if externally loaded, provided they have the same session number. Note that setting breakpoints in resident systems routines with globally shareable segments is not recommended. The option #<terminal> routes that task's console I/O to a remote terminal; otherwise, the task messages would appear on the SYMbug terminal. The option #* denotes no LUN's are to be passed to the task for terminal I/O. This option is required if the task is loaded external to SYMbug.

EXAMPLES		COMMENTS
SYMbug	? ATTA TEST	Attaches load module TEST to SYMbug.
SYMbug	? ATTA TEST,#CN01	Attaches load module TEST to SYMbug, but routes task messages to CN01.
SYMbug	? ATTA TEST,#*	Attaches TEST which was loaded external to SYMbug.

2.3.25 Detach a Task from SYMbug (DETA)

DETA [<task name>]

The DETA command disassociates that task from SYMbug. Execution of that task is not affected as it is known to and serviced by VERSAdos; however, SYMbug commands other than ATTA may not be issued to it. Existing breakpoints will be extracted automatically prior to detaching the named task. Detached tasks will remain in memory and are free to execute external to SYMbug control.

EXAMPLES		COMMENTS
SYMbug	? DETA	Detaches the foreground task.
SYMbug	? DETA SAMPLE	Detaches the task named SAMPLE.

2.3.26 Create an Event for a Task (EVEN)

```
EVEN [<task name>],<exception #>
```

The EVEN command will create an event of the exception number type specified for the named task. This is useful to allow checkout of Asynchronous Service Routine (ASR) related task code. An attempt to acknowledge the event with a AKQRST directive in the ASR will fail since no target task is waiting to resume execution.

EXAMPLES		COMMENTS
SYMbug	? EVEN,16	Stimulates a bus error for the foreground task.
SYMbug	? EVEN TEST,8	Queues a TRAP #8 event to the Asynchronous Service Queue (ASQ) of task TEST.

LOAD <file name> [<command line>]

The LOAD command is typically the first command issued in the multitasking mode. Standard VERSAdos file name conventions apply, with the default suffix being LO. If volume defaults apply, <file name> is simply <task name>. Any characters following the blank which terminates <file name> are assumed to be a command line which is passed appropriately in the VERSAdos initialized processor registers. Tasks are in the ready state after loading.

EXAMPLES		COMMENTS
SYMbug	? LOAD CART:0.XYZ.PROGRAM.LO	Loads the load module to memory.
SYMbug	? LOAD PROGRAM	Does the same assuming defaults apply.
SYMbug	? LOAD PROGRAM ;MAL	Loads the .LO module, passes the command line length in D6, and moves the command line text to the designated <command line> buffer.

MASK [<task name>],<exception #>

The MASK command inverts the bit of the specified exception in the named task's XM pseudo register. This will switch the enable/disable state of event recognition and the corresponding message display. For example, assuming the standard XM mask of $01FFFFF1, the command 'MASK , 2' will cause the foreground task to stop execution and display the message TRAP #2 PC=address whenever it calls the I/O handler. Issuing the MASK command again will reset XM from $01FFFFF3 to $01FFFFF1 and ignore TRAP #2 instructions in the user task.

EXAMPLES		COMMENTS
SYMbug	? MASK MAIN,7	Inverts the numbered bit in the XM register of a task named MAIN. This inverts (enables or disables) recognition of TRAP #7 events as breakpoints in the task named MAIN.
SYMbug	? MASK ,7	Does the same for the foreground task.

2.3.29 Start Execution of a Task (STAR) STAR

STAR [<task name>|ALL]

The STAR command commences execution of the named task if in the ready or wait for command state. This is equivalent to executing a GO command for the foreground task but without setting the OP register to zero. STAR ALL will continue the execution paths of all tasks not already in execution or in the dormant state.

EXAMPLES		COMMENTS
SYMbug	? STAR	Start execution of foreground task with prior OP option.
SYMbug	? STAR SAMPLE	Does the same for the task named SAMPLE.
SYMbug	? STAR ALL	Starts all tasks in ready or wait state.

STAT [<task name>,<status>]

where <status> may be:

DORM issues STOP directive to a ready task. Task becomes dormant.

REDY issues START directive to a dormant task. Task becomes ready for execution.

WAKE issues WAKEUP directive to a waiting task. Task becomes ready for execution.

The STAT command lists the status information of all tasks or allows a specified task's status to be changed.

The STATUS display header includes these fields:

TASK	First four characters of the task name.
SESS	Four digit session number.
STATE	Literal status REDY, WAIT, or DORM prefixed by 'e' if in execution.
EVENT	Contains 4 subfields of data < = foreground task Task Note Level = the lead digit (see TASK command) Last Event Type = A,D, or X (Attach, Detach, or Exception) Last Event Code = 2 hex digits (22 = Breakpoint)
@PC	Task PC following last event.
PC NOW	PC (NOW being a few milliseconds ago if task in execution).
SR	Status Register in hex (also a few milliseconds ago).
MASK	Current XM pseudo register in effect for task event exceptions.
TCB STAT	Hex long word. Consult RSTATE directive for bit interpretation. Refer to VERSAdos Data Management Services and Program Loader User's Manual, RMS68KIO.
OP	Current OP pseudo register in effect for task's execution options.
CRT	Terminal id assigned to task for normal keyboard and screen I/O.

EXAMPLES		COMMENTS
SYMbug	? STAT	Displays status of all tasks known to SYMbug.
SYMbug	? STAT ABCD,REDY	Changes task ABCD's state to REDY.

2.3.31 Stop Execution of a Task (STOP) STOP

STOP [<task name>|ALL]

The STOP command stops execution of a task and leaves it in the ready state. A STOPPED event message for the task signals completion of the process. STOP ALL will affect all tasks currently flagged in execution. Note that the BREAK key has no effect on task's execution status.

EXAMPLES		COMMENTS
SYMbug	? STOP	Stops execution of the foreground task.
SYMbug	? STOP CHARLIE	Stops execution of the task named CHARLIE.
SYMbug	? STOP ALL	Stops execution of all tasks running under SYMbug.

TASK <task name>[,<note level>]

The named task will appear in the next prompt as the foreground task. Primitive level SYMbug commands now apply to that task. The note level option directs SYMbug's response to breakpoints for this task. The following codes allow the user to see or suppress breakpoint messages as well as to halt or continue execution as a result of encountering a breakpoint.

Note Level	Task Execution	Breakpoint Message	Become Foreground	Use
0	STOPS	Suppressed	No	Concentrate on foreground
1 (default)	STOPS	Displayed	No	Normal setting
2	STOPS	Displayed	Yes	Saves TASK command
3	CONTINUES	Displayed	No	Procedure trace without GO's
4	CONTINUES	Displayed	No	Disables breakpoints without reentering

EXAMPLES		COMMENTS
SYMbug	? TASK FILLY	Change prompt to new foreground task named FILLY.
SYMbug	? TASK METOO	Suppresses breakpoint messages for task named METOO.

TERM <task name>

This command banishes a task from memory. Execution terminates and the task becomes unknown both to SYMbug and VERSAdos. The LOAD command would reacquaint it with VERSAdos, and ATTA would then establish SYMbug control over it. SYMbug does not automatically terminate itself when all user tasks are terminated. The QUIT command will terminate all user tasks and the SYMbug task. Use QUIT rather than TERM to conclude a debugging session.

EXAMPLE		COMMENT
SYMbug	? TERM DEANNA	Terminate the user task named DEANNA.

2.3.34 Wait for Event (WAIT)

WAIT

The WAIT command will suppress the SYMbug prompt message until a task has an exception event or the BREAK key is depressed. Task-directed I/O to the CRT is unaffected.

EXAMPLE		COMMENT
SYMbug	? WAIT	SYMbug will cease prompting for commands.

CHAPTER 3

DEbug COMMANDS

3.1 INTRODUCTION

This chapter explains how to invoke the DEbug prompt, describes the command format structure, and provides a detailed explanation of the primitive commands.

3.2 INVOKING THE DEbug PROMPT

In response to the VERSAdos prompt (=), the user enters one of the following:

=DEBUG	Invokes the multitasking mode.

after which the following is displayed:

```
Debug:Revision x.xx
 MAXIMUM TASK COUNT (1->19)?
```

or

=DEBUG TESTPROG	Monitors the named task with an optional command line in single task mode.

after which the following is displayed:

```
Debug:Revision x.xx
Debug TEST ATTACHED      PC=00001000
Debug TEST?
```

3.2.1 DEbug Messages

The following messages may be encountered during the execution of DEbug.

TABLE 3-1. DEbug Messages

MESSAGE	EXPLANATION
Debug <task> ?	<task> is the first four letters of the foreground task name. The user may respond with any primitive or task level DEbug command.
Debug ?	Is the initial prompt in multitask mode. The LOAD, ATTA, HELP, and QUIT are the only legal commands until a foreground task is declared.
Debug <task> WHAT ?	A bell will also ring to signify a syntax error in the previous DEbug command. Consult the HELP command listing.
Debug <task> ATTACHED PC=oooooooo.	The PC contains the address of the instruction following the instruction which caused the event.

The following exception event messages will be displayed provided the corresponding bit in the exception mask is set.

BIT NUMBER	MESSAGE	
0	Not used	
1-15	TRAP #1 -> TRAP #15	PC=xxxxxxxx
16	BUS ERROR	"
17	ADDRESS ERROR	"
18	ILLEGAL INST	"
19	ZERO DIVIDE	"
20	CHK INST	"
21	TRAPV	"
22	PRIV VIOLATION	"
23	LINE 1010	"
24	LINE 1111	"
25-31	Not used	
Unmaskable Events	BREAKPOINT	PC=xxxxxxxx
	ATTACHED	"
	DETACHED	"
	STOPPED	"
	TRACE ONE INST	"
	TRACE MAX INST	"
	VALUE CHANGE	"
	VALUE EQUAL	"

These event messages are queued to tasks as they occur and will not be displayed until the current DEbug command has been processed and a new prompt is about to be displayed.

3.2.2 Monitoring Execution of a User Task

No single task may have multiple paths of execution; however, all user tasks may be simultaneously active under DEbug control. There are five methods to initiate a path of execution for a task.

For explanation of OP and XM pseudo registers, refer to paragraph 3.2.3.

a. GO - starts the foreground task with OP=0000

b. TR - starts the foreground task with OP=1000 or 0800

c. AS - starts the foreground task with OP=2000 or 3000 and XM=FFFFFFFF

d. STAR <task name> - starts the named task with OP unchanged

e. STAR ALL - starts all ready or waiting tasks with OP unchanged

Once set into execution, a task will continue to execute until it:

a. Has a normal (maskable) exception that is enabled by its exception mask.

b. Has an unmaskable exception event controlled by that task's pseudo registers, including tracing a specified number of instructions, changing the monitored address, or hitting a breakpoint.

c. Is explicitly stopped by the STOP, TERM, or QUIT command.

d. Terminated normally or abnormally via a TRAP #1 directive under program control.

NOTE: The BREAK key will NOT affect the execution of any task.

In single tasking mode, the DEbug prompt is delayed until the event which concludes execution has occurred. In multitasking mode, the DEbug prompt is returned to the user prior to the completion of a task's execution. The user may now issue DEbug commands to other tasks or type WAIT. In either case, an event message will notify the user when the executing task has concluded.

The STAT command will permit a snapshot of a task's progress while in execution. An 'e' displayed before a task's status indicates that the task is actively executing. A DF command will then display the register values of the associated task at the time of the last STAT command. Changes to a task's registers, pseudo registers, or breakpoint addresses are prohibited during execution. To set a new breakpoint, it is necessary to use the command sequence:

Debug	? STOP <task name>	Resets XM to 01FFFFF1.
Debug	? TASK <task name>	Makes <task name> the foreground task.
Debug	? BR <address>	Sets the new breakpoint.
Debug	? STAR <task name>	Allows previous .OP register options to remain in effect.

The MD command may be used to display a task's memory during execution. MS is also enabled, but extreme caution is urged in regards to dynamically altering an executing task's memory.

3.2.3 DEbug Pseudo Registers

In addition to the normal task processor registers A0-A7, D0-D7, PC, and SR, the user task can manipulate the following pseudo registers which communicate specific constraints to VERSAdos in controlling the user task's monitored execution.

MC The maximum count value controls the count of user task instructions to be traced during execution. Displayed as a 4-byte value, the low order two bytes are the count while the upper bytes are updated by VERSAdos to reflect the current count of instructions traced to date. MC is in effect if OP bit 11 is set. Thus OP=2800 combines the address stop and maximum instruction count features. See .OP, .MC, and TR.

OP The execution option pseudo register controls the monitored manner of execution. It permits the user task to run free, trace a specified number of instructions, or monitor a given memory address for change in conjunction with the settings of the pseudo registers MC, VA, VL, VM, and XM. The need to adjust these pseudo registers explicity will be minimized by GO, TR, and AS, which provide the most common settings automatically. See .OP, .MC, .VA, .VL, GO, TR, AS, and STAT.

VA The value pseudo register contains a 4-byte value masked by the VM pseudo register for comparison with the current contents at the memory address contained in VL. In the address stop on equal mode setting of OP (=3000), the VALUE EQUAL event will stop the task's execution with the PC at the instruction following that which set the monitored location to the VA value. See .VA, .VL, .VM, and .OP.

VL The value location pseudo register is a 4-byte even address of the memory location within the task's segment to be monitored for change in conjunction with OP=(2000 or 3000). See .VA, .VL, .OP, and AS.

VM The value mask is a 4-byte mask ANDed to the memory content at address location (VL) prior to comparison with the value (VA). Under OP=3000, a VALUE EQUAL event will occur appropriately. Note that a mask setting of FF000000 will isolate a single byte for comparison.

XM The exception monitor mask controls the display of exception event messages declared previously. The mask initially defaults to 01FFFFF1, which enables all events except TRAPS #1, #2, and #3. If a user task desired to handle its own zero divide exceptions, a mask value of 01F7FFF1 would apply. See .XM, MASK, and STAT instructions to manipulate XM.

3.3 PRIMITIVE COMMANDS

DEbug primitive commands are listed in the following table.

TABLE 3-2. DEbug Primitive Commands

COMMAND SYNTAX	DESCRIPTION
AS [<address>] [<value>]	Address stop
[NO]BR [<address>]...	Set/reset breakpoint
DE	Default to attach/detach printer
DF	Display format
G[O]	Execute target task
HE[LP]	Display commands
MD <address> [<count>]	Memory display
MS <address> <byte 1> [<byte 2> <byte 3>]...	Memory set
OF [<register> <value>]	Offset
Q[UIT]	Quit (go to VERSAdos)
T[R] [<count>]	Trace target task
ATTA <task name>[,<terminal>|#*]	Attach task
DETA [<task name>]	Detach task
EVEN [<task name>],<exception #>	Event definition
LOAD <file name> [<command line>]	Load (task)
MASK [<task name>],<exception #>	Mask exception
STAR [<task name>]|ALL]	Start task(s)
STAT [<task name>,<status>]	Status definition
STOP [<task name>|ALL]	Stop task(s)
TASK <task name>[,<note level>]	Task notify
TERM <task name>	Terminate task
WAIT	Wait task
.A0-.A7	Display/change address register
.D0-.D7	Display/change data register
.MC	Display/change maximum count (software register)
.OP	Display/change execution options (software register)
.PC	Display/change program counter
.SR	Display/change status register
.ST	Display/change task state
.VA	Display/change value (software register)
.VL	Display/change value location (software register)
.VM	Display/change value mask (software register)
.XM	Display/change exception mask
BREAK	Abort command
CTRL-S	Redisplay line
CTRL-H	Delete character
CTRL-W	Suspend output (1)
CTRL-X	Cancel command line
CR (Carriage Return)	Send line for execution

NOTE: (1) When CTRL-W is used, the entry of any character will cause the output display to continue.

3.3.1 Address Stop (AS)

AS [<address>] [<value>]

The AS command starts program execution with a breakpoint type condition on the designated value address (.VA) pseudo register. The values of value location (.VL), value mask (.VM), and .VA are updated, depending on parameters supplied. If both address and value parameters are provided, program execution will terminate when the address equals value. Otherwise, a stop-on-address-change rather than stop-on-equal mode of execution is in effect. The .VM register is applied to memory location .VL to achieve comparisons of less than four bytes. Thus, .VM values of $FF000000 and $FFFF0000 are used for byte or word fields beginning at location .VL.

COMMAND FORMAT	DESCRIPTION
AS	Target begins execution using previous values of .VL, .VA, and .VM in stop-on-address-change mode.
AS <address>	<address> is stored in .VL and target execution begins in stop-on-change mode.
AS <address> <value>	Both .VL and .VA are updated and target execution commences in stop-on-equal mode.
AS <address> <value>;<mask>	Same as preceding format except <mask> specifies B, W, or L. Default <mask> is L.
AS <address>;<mask>	Same as second example above except .VM is updated to a <mask> of B, W, or L.

EXAMPLES		COMMENTS
Debug	? AS 1D08	$1D08 is stored in .VL. Task begins execution at address in .PC. $2000 is stored in .OP to cause memory location $001D08 to be compared with .VA for change after executing each instruction. .VA is initialized by the monitor as the original content of $001D08 prior to executing the first instruction. This is stop-on-address-change mode.
Debug	? AS 1D08 20	Functions as above except a value of $FF000000 is placed in .VM and $20000000 is placed in .VA. An exception event to halt the task will occur only if program execution caused the byte at address $001D08 to become $20. This is stop-on-address equal mode.

3.3.2 Set Breakpoints (BR) BR
Reset Breakpoints (NOBR) NOBR

```
BR [<address>]...
NOBR [<address>]...
```

The BR command enters the address into the internal breakpoint table. During execution of the target task, a debug halt occurs immediately preceding the execution of any instruction whose address is in the breakpoint table. In this case, the halt reason is breakpoint.

The NOBR command is used to remove one or more breakpoints from the internal breakpoint table, and functions as the inverse of the BR command.

COMMAND FORMAT	DESCRIPTION
BR	Display all current breakpoints.
BR <address>	Set a breakpoint.
BR <address1> <address2>...	Set several breakpoints.

NOTE

DEbug affects breakpoints by implanting a privileged instruction at the breakpoint addresses. Thus, breakpoints will not be properly processed unless bit 22 is set in the exception mask.

COMMAND FORMAT	DESCRIPTION
NOBR	Reset all current breakpoints.
NOBR <address>	Reset a breakpoint.
NOBR <address1> <address2>...	Reset several breakpoints.

EXAMPLES		COMMENTS
Debug	? BR 1080	Sets a breakpoint at $1080.
Debug	? BR 1084 109A 984	Sets three breakpoints.
Debug	? NOBR 109A	Removes the third breakpoint.
Debug	? NOBR	Removes the remaining breakpoints.

3.3.3 Default to Attach/Detach Printer (DE) DE

DE

The DE command displays the current state of the printer echo option. Responding Y or N to the prompt will cause MD output to be echoed to the line printer or suppressed.

EXAMPLE	COMMENT
Debug ? DE PR ECHO = N (Y/N)? Y	Enables the printer.

3.3.4 Display Target Task Registers (DF) DF

DF

The DF command is used to display all registers of the foreground task.

COMMAND FORMAT	DESCRIPTION
DF	The contents of the following target task registers are displayed: D0-D7 A0-A7 PC SR

EXAMPLES	COMMENTS
Debug ? DF	Displays the registers as follows:

```
D0-D7 00000000 00000000 00000000 00000000...
A0-A7 00000000 00000000 00000000 00000000...
 PC=00000000 SR=0000
```

3.3.5 Execute Target Task (G) G

G[O]

The G or GO command is used to cause execution of the foreground task. The option (.OP) pseudo register is automatically set to $0000.

EXAMPLE		COMMENT
Debug	? G	Target begins or continues to execute.

3.3.6 Display Commands (HE) HE

HE[LP]

The HE or HELP command lists the primitive commands, as shown below:

```
     AS     [<addr>] [<value>]          ADDRESS STOP (On change or = Value)
[NO] BR     [<addr>]....                BREAKPOINT List, Set or Delete Multiple
     DE                                 DEFAULT List, Set Options
     DF                                 DISPLAY FORMATTED Registers
     G      [<addr>]                    GO into Execution (Foreground Task)
     MU     <addr> [<count>]            MEMORY DISPLAY (default count=16 bytes)
     MS     <addr> <data>....           MEMORY SET (Data bytes spaced out)
     OF     [<reg> <value>]             OFFSET REGISTER List, Set or Delete
     QUIT                               QUIT (Terminate Debugging Session)
     TR     [count]                     TRACE      (CR will continue TRACE)
     .A0->.A7, .D0->.D7 .PC .SR .XM .ST .VL .VA .VM .MC  REGISTER List or Set
     ATTA   [<task>] [<#crt>]           ATTACH Task to DEBUG (Remote crt I/O)
     DETA   [<task>]                    DETACK Task from DEBUG (Continues Exec)
     EVEN   [<task>] <#exception>       EVENT CREATED for a Task
     LOAD   <task> [<comline>]          LOAD TASK (with COMLINE)
     MASK   [<task>] <#exception>       MASK Exception Toggled
     STAR   [<task> | ALL]              START Task(s) into Execution like GO
     STAT   [<task> <status>]           STATUS List or Set (DORM,REDY or WAKE)
     STOP   [<task> | ALL]              STOP Task(s) Execution & Set DORMANT
     TASK   <task> [<#notify>]          TASK becomes FOREGROUND with BP option
     TERM   <task>                      TERMINATE Task
     WAIT                               WAIT for {BREAK} to display PROMPT

Debug      ?
```

3.3.7 Memory Display at Terminal (MD)

MD <address> [<count>]

The MD command is used to display the count bytes of memory beginning at <address> on the user's terminal. The displayed bytes must be contained in one segment.

COMMAND FORMAT	DESCRIPTION
MD <address>	Display memory beginning at <address> for a count of $10 bytes.
MD <address> <count>	Display memory beginning at <address> for count bytes.

EXAMPLES		COMMENTS
Debug	? MD 400	Displays 16 bytes beginning at address 00000400.
Debug	? MD 400 30	Displays memory content from 400 through 42F.
Debug	? MD 0 200	512 bytes of memory displayed beginning at address 00000000.

MS <address> <bytel [<byte2 <byte3]...

The MS command is used to set memory beginning at <address> and extending through subsequent locations to the values specified by the input bytes.

COMMAND FORMAT	DESCRIPTION
MS <address> <byte 1>	Set the byte at <address> to <byte 1>.
MS <address> <byte 1> <byte 2>	Set the bytes at <address> and <address+1> to <byte 1> and <byte 2>, respectively. A space is required between multiple byte entries.

EXAMPLES		COMMENTS
Debug	? MS 400 0A	Sets memory address 000400 to $0A.
Debug	? MS 401 0B 0C 0D	Sets addresses 401-403 to $0B, $0C, and $0D, respectively.

3.3.9 Base Register Offsets (OF)

OF [<register> <value>]

The OF command is used to define up to eight base registers numbered 0-7 within the task that corresponds with section definitions that reset the relative location counter to zero. This permits syntax like 6+R3 to be evaluated as logical address $1A06 if base register 3 was initialized to $1A00. If base registers are defined they will be used in displaying memory or breakpoint addresses.

COMMAND FORMAT	DESCRIPTION
OF	Display current base register values
OF <register> <value>	Define base register number (register) to be that hex value.
OF <register> 0	Cancel effective base register.

EXAMPLES		COMMENTS
Debug	? OF	Displays the eight offset base registers.
Debug	? OF 1 4AA	Sets R1 to 000004AA.
Debug	? OF 1 0	Resets R1 to zero.

3.3.10 Terminate Debugging Session (Q) Q

Q[UIT]

The Q or QUIT command is used to terminate DEbug and all target tasks simultaneously.

EXAMPLE	COMMENT
Debug ? Q[UIT] =	DEbug immediately terminates all target tasks and returns control to VERSAdos.

3.3.11 Trace Target Task (T)

T[R] [<count>]

The T or TR command causes execution of a specified number of instructions. If <count> is omitted, the default is one. In trace one mode, a carriage return will trace one more instruction. The PC is displayed with the event message of the next instruction to be executed. In trace count mode, TRAP instructions are not counted.

COMMAND FORMAT	DESCRIPTION
T	Traces one instruction.
T <count>	Traces number of instructions specified by count.

EXAMPLES		COMMENTS
Debug	? TR	Traces the next instruction. A (CR) will trace one more instruction each time it is depressed.
Debug	? TR 6	Stores 6 in .MC register. Sets .OP register to $0800, which results in tracing the next six instructions.

3.3.12 Attach a Task to DEbug (ATTA) ATTA

ATTA <task name>[#<terminal>|#*]

The ATTA command will attach DEbug to a task already in memory. Tasks usually are first loaded using the LOAD command; however, they may be attached to DEbug if externally loaded as long as they have the same session number. The option #<terminal> routes that task's console I/O to a remote terminal; otherwise, the task messages will appear on the DEbug terminal. The option #* denotes no LUNS are to be passed to the task for terminal I/O. This option is required when the task was loaded external to DEbug.

EXAMPLES		COMMENTS
Debug	? ATTA ,#CN01	Routes task messages to CN01.
Debug	? ATTA TEST	Attaches load module TEST to DEbug.
Debug	? ATTA TEST,#*	Attaches TEST which was loaded externally to DEbug.

3.3.13 Detach a Task from DEbug (DETA)

DETA [<task name>]

The DETA command disassociates that task from DEbug. Execution of that task is not affected as it is known to and serviced by VERSAdos; however, DEbug commands other than ATTA may not be issued to it. Existing breakpoints will be extracted automatically prior to detaching the named task.

EXAMPLES		COMMENTS
Debug	? DETA	Detaches the foreground task.
Debug	? DETA MASTER	Detaches the task named MASTER.

3.3.14 Create an Event for a Task (EVEN)

EVEN [<task name>],<exception #>

The EVEN command will create an event of the exception number type specified for the named task. This is useful to allow checkout of ASR-related (Asynchronous Service Routine) task code. An attempt to acknowledge the event with an AKQRST directive in the ASR will fail since no target task is waiting to resume execution.

EXAMPLES		COMMENTS
Debug	? EVEN ,16	Simulates a bus error for the foreground task.
Debug	? EVEN TEST,8	Queues a TRAP #8 event to the ASQ (Asynchronous Service Queue) of task TEST.

LOAD <file name> [<command line>]

The LOAD command is typically the first command issued in the multitasking mode. Standard VERSAdos file name conventions apply with the default suffix being .LO. If volume defaults apply, the first four characters of <file name> are used as <task name>. Any characters following the blank which terminates <file name> are assumed to be a command line which is passed appropriately in the VERSAdos initialized processor registers. Tasks are in the ready state after loading.

COMMAND FORMAT	DESCRIPTION
LOAD <volume>:<user#>.<catalog>.<file name>.<ext>	Loads the load module into memory.
LOAD <file name>	Does the same, assuming defaults apply.
LOAD <file name> <command line>	Loads the .LO module, passes the command line length in D6, and moves the command line text to the designated <command line> buffer.

EXAMPLE	COMMENT
Debug ? LOAD TESTASM ;D	A test version of an assembler is loaded with the ;D option. The <task name> is TEST.

MASK [<task name>],<exception #>

The MASK command inverts the bit of the specified exception in the named task's XM pseudo register. This will switch the enable/disable state of event recognition and the corresponding message display. For example, assuming the standard XM mask of $01FFFFF1, the command 'MASK , 2' will cause the foreground task to stop execution and display the message 'TRAP #2 PC=address' whenever it calls the I/O handler. Issuing the MASK command again will reset XM from $01FFFFF3 to $01FFFFF1 and ignore TRAP #2 instructions in the user task.

EXAMPLES		COMMENTS
Debug	? MASK MAIN,7	Inverts the numbered bit in the named task's XM register.
Debug	? MASK ,7	Does the same for the foreground task.

3.3.17 Start Execution of a Task (STAR)

STAR [<task name>|ALL]

The STAR command commences execution of the named task if in the ready or wait command state. This is equivalent to doing a GO command for the foreground task but without setting the .OP register to zero. STAR ALL will continue the execution paths of all tasks not already in execution or in the dormant state.

EXAMPLES		COMMENTS
Debug	? STAR	Start execution of foreground task with prior OP option.
Debug	? STAR SAMPLE	Does the same for the task named SAMPLE.
Debug	? STAR ALL	Starts all tasks in ready or wait state.

STAT [<task name>,<status>]

where <status> may be:

DORM issues STOP directive to a ready task. Task becomes dormant.

REDY issues START directive to a dormant task. Task becomes ready for execution.

WAKE issues WAKEUP directive to a waiting task. Task becomes ready for execution.

The STAT command lists the status information of all tasks or allows a specified task's status to be changed.

The STATUS display header includes these fields:

TASK	First four characters of the task name.
SESS	Four digit session number.
STATE	Literal status REDY, WAIT or DORM prefixed by "e" if in execution.
EVENT	Contains 4 subfields of data < = the foreground task Task Note Level = the lead digit (see TASK command) Last Event Type = A, D, or X (Attach, Detach, Exception) Last Event Code = 2 hex digits (22 = Breakpoint)
@PC	Task PC following last event.
PC NOW	PC (being a few milliseconds prior if task in execution).
SR	Status register in hex (a few milliseconds prior).
MASK	Current .XM pseudo register in effect for task event exceptions.
TCB STAT	Hex long word. Consult RSTATE directive for bit interpretation.
OP	Current .OP pseudo register in effect for tasks execution options.
CRT	Terminal ID assigned to task for normal keyboard and screen I/O.

EXAMPLES		COMMENTS
Debug	? STAT	Displays status of all tasks known to DEbug.
Debug	? STAT DORM,REDY	Changes task DORM's state to REDY.

3.3.19 Stop Execution of a Task (STOP)

STOP

STOP [<task name>]|ALL

The STOP command terminates execution of a task and leaves it in the ready state. A STOPPED event message for the task signals completion of the process. STOP ALL will affect all tasks currently flagged in execution. Note that the BREAK key has no effect on task's execution status.

EXAMPLES		COMMENTS
Debug	? STOP	Stops execution of the foreground task.
Debug	? STOP WRITER	Stops execution of the task named WRITER.
Debug	? STOP ALL	Stops execution of all tasks running under DEbug.

TASK [<task name>][,<note level>]

The named task will appear in the next prompt as the foreground task. Primitive level DEbug commands now apply to that task. The <note level> option directs DEbug's response to breakpoints for this task. The following codes allow the user to see or suppress breakpoint messages, as well as to halt or continue execution as a result of encountering a breakpoint.

Note Level	Task Execution	Breakpoint Message	Become Foreground	Use
0	STOPS	Suppressed	No	Concentrate on foreground
1 (default)	STOPS	Displayed	No	Normal setting
2	CONTINUES	Displayed	Yes	Saves TASK command
3	CONTINUES	Displayed	No	Procedure trace without GO's
4	CONTINUES	Suppressed	No	Disables breakpoints without reentering

EXAMPLES		COMMENTS
Debug	? TASK IMPACT	Change prompt to new foreground task named IMPACT.
Debug	? TASK SHORT,0	Suppresses breakpoint messages for task named SHORT.
Debug	? TASK ,2	Current task becomes foreground when a breakpoint is encountered.

3.3.21 Terminate a Task's Execution (TERM)

TERM

TERM <task name>

The TERM command banishes a task from memory. Execution terminates and the task becomes unknown both to DEbug and VERSAdos. The LOAD command would reacquaint it with VERSAdos and ATTA would then establish DEbug control over it. DEbug does not automatically terminate itself when all user tasks are terminated. The QUIT command will terminate all user tasks and the DEbug task. Use QUIT rather than TERM to conclude a debugging session. <task name> is a required argument.

EXAMPLE		COMMENT
Debug	? TERM CURE	Terminate the user task named CURE.

3.3.22 Wait for Event (WAIT)

WAIT

The WAIT command will suppress the DEbug prompt message until a task has an exception event or the BREAK key is depressed. Task directed I/O to the CRT is unaffected.

EXAMPLE	COMMENT
Debug ? WAIT	DEbug will cease prompting for commands.

3.3.23 Display/Change Specified Register

.<register> [<value>]

The .<register> command is used to display or set the specified register or pseudo register belonging to the foreground task.

.A0-.A7	address register
.D0-.D7	data register
.MC	maximum instruction count (software register)
.OP	option (software register)
.PC	program counter register
.SR	status register
.ST	task state register
.VA	value (software register)
.VL	value location (software register)
.VM	value mask (software register)
.XM	exception mask register

COMMAND FORMAT	DESCRIPTION
.<register>	Display contents of specified register.
.<register> [<value>]	Replace contents of specified register with value.

See also: DF, MD, OF.

EXAMPLES		COMMENTS
Debug	? .A5	Displays address register A5.
Debug	? .A5 3FD	Sets address register A5 to $3FD.
Debug	? .D4 0	Zeros data register D4.
Debug	? .PC	Displays the program counter address at which execution of the foreground task will resume following a GO, TR, STAR, or AS command.
Debug	? .PC 1020	Sets the program counter to resume execution at memory location $1020.
Debug	? .MC 4	Sets maximum instruction count for trace at 4.
Debug	? .OP	Displays current options register for monitor: $0000 means GO $0800 means TRACE maximum <count> instructions. $1000 means TRACE one instruction. $2000 means STOP on .VL address change. $3000 means STOP on .VL address equal .VA.

Debug	? .SR	Displays the status register condition codes. Bits 0-7 are user byte; bits 8-15 are system byte. User byte format is 000XNZVC for extend, byte negative, zero, overflow, and carry conditions, as set by the previously-executed instruction.
Debug	? .SR 8004	Sets trace mode (system byte) and zero condition in status register.
Debug	? .ST	Displays monitor status word of task. See RSTATE directive for bit interpretation.
Debug	? .VA 5	Sets value pseudo register to 5.
Debug	? .VL 10C8	Sets value location (to be monitored by stop-on-address option) to $10C8.
Debug	? .VM	Displays value mask applied to the content of memory at .VL before comparison with value in .VA.
Debug	? .XM 01FFFF	Sets exception mask register to enable all events except TRAPS #1, #2, #3, and #7.

SUGGESTION/PROBLEM REPORT

MICROSYSTEMS

QUALITY • PEOPLE • PERFORMANCE

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: _____ Manual: _____

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please Print

Name _____ Title _____

Company _____ Division _____

Street _____ Mail Drop _____ Phone _____

City _____ State _____ Zip _____

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Microsystems Field Service Support
(800) 528-1908
(602) 829-3100

MOTOROLA

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

16362-1 PRINTED IN USA (11/83) MPS 2M